VLSI Design with Electric

A Tutorial By

David Harris
Harvey Mudd College

July 19, 2001

1. Introduction

The Information Age is made possible by the incredible ability to pack vast numbers of circuits onto inexpensive integrated circuits, or chips. These Very Large Scale Integration (VLSI) chips can contain many millions of transistors. Computer-aided design tools are necessary to layout and verify such complex chips. This tutorial assumes a background in digital electronics, particularly in combinational circuit design. At the end of the tutorial you will be able to use the Electric CAD suite to design digital circuits at the transistor level, layout your circuits, verify your design, and send your completed chip for manufacturing.

The tutorial is divided into four sections. First we explore building logic gates from transistors. Then we use Electric to draw such transistor-level schematics of gates and simulate the circuits. Next we describe how transistors are fabricated and represented with layout. Finally we use Electric to layout logic gates and verify the layout.

2. Transistors and Logic Gates

Transistors are the fundamental building blocks of VLSI systems. This section shows how to think of transistors as electrically controlled switches and how to assemble transistors into more complex logic gates.

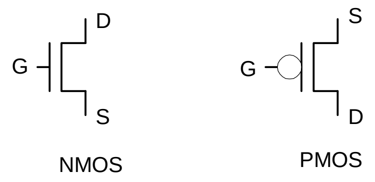
2.1. CMOS Transistors

Transistors are just electrically controlled switches. A transistor has three terminals. Depending on the state of the control terminal, the other two signal terminals may be connected or disconnected. There are many different types of transistors built with different materials and structures. We will focus on Complementary Metal[1]-Oxide-Semiconductor (CMOS) transistors.

There are two types of CMOS transistors: n-type Metal-Oxide-Semiconductor (NMOS) and p-type MOS (PMOS). Circuits using both types are called CMOS. CMOS

[1] CMOS transistors don’t use metal anymore, but they once did and the name stuck!

transistors have three terminals: the gate, source, and drain. The source and drain are interchangeable.



When the gate of an NMOS transistor is high, we say the transistor is ON and there is a conducting path from source to drain. When the gate is low, we say the transistor is OFF and there is no path from source to drain. A PMOS transistor is just the opposite, as the bubble on the gate might hint. When the gate is low, the transistor is ON. When the gate is high, the transistor is OFF.

2.2. The CMOS Inverter

We can build an inverter out of an NMOS transistor and a PMOS transistor. The input A controls the gates of both transistors. The output Y is connected to the drains of both transistors. The source of the NMOS transistor is attached to ground (GND) and the source of the PMOS transistor is attached to power (VDD), i.e. 5 volts. Note that by convention, we call the side closer to the rail (VDD/GND) the source and the side closer to the output the drain even though the source and drain are symmetric.

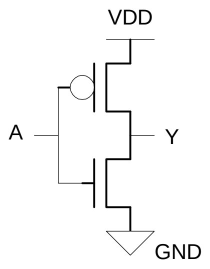


Figure 1: Inverter schematic

When A is 1, the NMOS transistor is ON and the PMOS transistor is OFF. Therefore, the output Y has a conducting path to GND through the NMOS transistor, but no path to VDD, so it is pulled down to 0. When A is 0, the PMOS transistor is ON and the NMOS transistor is off. Therefore the output is pulled up to 1 through the PMOS transistor. Thus Y = ~A.

A	NMOS	PMOS	Y
0	OFF	ON	1
1	ON	OFF	0

2.3. Other Logic Gates

Let's design some other logic gates. We can fill out a truth table for the following circuit. As laziness is an engineer's virtue, we do not label VDD and GND explicitly; the flat line up top indicates VDD and the triangle indicates GND. When both inputs are 1, both NMOS transistors turn on and we have a path to pull Y low, i.e. to logic 0. When either input is 0, at least one of the PMOS transistors turn on and we have a path to pull Y high, i.e. to logic 1. One of the series NMOS transistors is off, so there is no path fighting down to ground. When both inputs are 0, both PMOS transistors work together to pull Y high. From the truth table, we see this is a NAND gate.

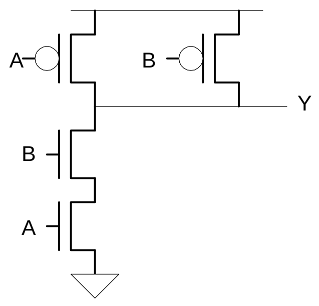


Figure 2: 2-input NAND gate schematic

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

A NOR gate is similar but uses series PMOS devices and parallel NMOS devices. When either input is true, the output is pulled down low through at least one of the NMOS transistors. When both inputs are false, the series PMOS transistors are both ON to pull the output high.

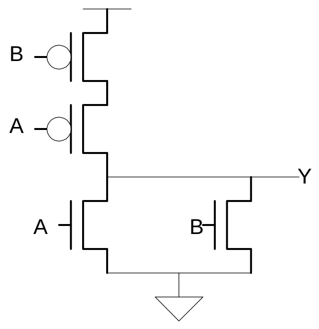


Figure 3: 2-input NOR gate schematic

A	B	Y
0	0	1
0	1	0

1	0	0
1	1	0

In general, logic gates are constructed with a network of NMOS transistors between the output Y and GND and another network of PMOS transistors between the output and VDD. As a specific example, how would we build a 3-input NAND? The output should only be low when all three inputs are high, so we want three NMOS transistors in series. The output should pull high when any of the inputs is low, so we want three PMOS transistors in parallel.

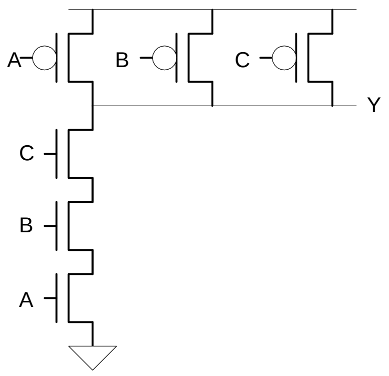


Figure 4: 3-input NAND gate schematic

It is sufficient to design the NMOS network that determines when the output should be low. The PMOS network is formed by taking the "conduction complement." This involves replacing all series transistors with parallel transistors and parallel transistors with series transistors. This guarantees that the output is always pulled up to VDD or pulled down to GND. If it were pulled both directions simultaneously, we would have a short circuit. The chip might get warm, and in the worst case might let out the magic smoke that makes it work! If the output is not pulled up or down, we say it is floating. A floating node is neither known to be high nor low. It may tend to hold its old value for a few milliseconds, but may leak to an intermediate value between 0 and 1.

2.4. Noninverting Gates

So far, we have only designed inverting gates such as NOT, NAND, or NOR. How do we build noninverting gates such as AND or OR? It is impossible using the approach we've tried so far with NMOS transistors pulling down because that causes the output to tend to go low as more inputs are high. We might try using NMOS transistors to pull up to VDD and PMOS to pull down to GND. Unfortunately, transistors are not perfect switches. NMOS are best at pulling down and PMOS are best at pulling up. Reversing the use leads to a poor gate.

Instead, we build noninverting gates out of inverting gates followed by inverters. For example, a 2-input AND is made from a 2-input NAND driving an inverter.

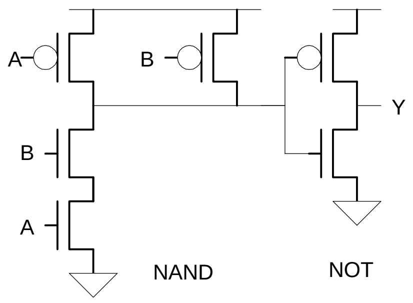


Figure 5: 2-input AND gate schematic

2.5. Tristate Inverter

The next circuit violates our conduction complement rule. It is called a tristate inverter and has the following truth table and symbol.

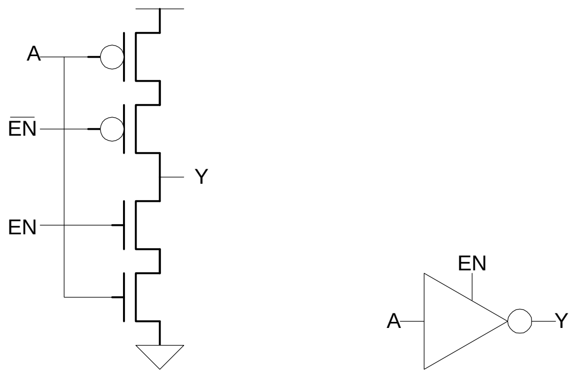


Figure 6: Tristate inverter schematic and symbol

EN	A	Y
0	0	floating
0	1	floating
1	0	1
1	1	0

The tristate gets its name because the output may be driven to one of three states: 0, 1, or floating. When the enable EN is 1, the tristate acts like a regular inverter. When EN is 0, the output floats. Why would we ever want an output to float? In general, we don't. However, we can use two tristates working in parallel to construct useful circuits.

2.6. Multiplexer

One useful circuit is called a multiplexer or mux. A mux selects one of two inputs D0 and D1 depending on the value of the select S. An inverting mux can be built from two tristates. Notice how the outputs of the two tristates are shorted together. Usually this would be a bad thing. However, in this design, the enable signals to the tristates are complementary. Therefore, exactly one will be on and one will be off at any given time. The output will never float and will never be driven to two different values simultaneously. We could build a noninverting mux by adding an inverter to the output.

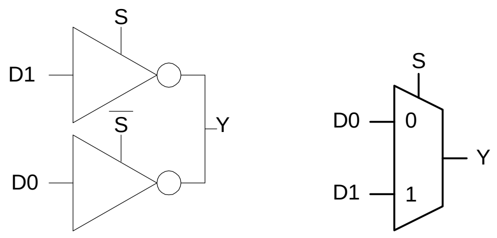


Figure 7: Multiplexer schematic and symbol

S	D1	D0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

2.7. Latch

Another useful circuit is the transparent latch. Recall that a latch has a data and a clock input. When the clock G is high, we say the latch is transparent and the data passes through to the output. When the clock is low, we say the latch is opaque and the output retains its old value. We could build a latch like we did out of ordinary logic gates. The design would require 22 transistors: 2 for the inverter, 4 for each NOR, and 6 for each NAND.

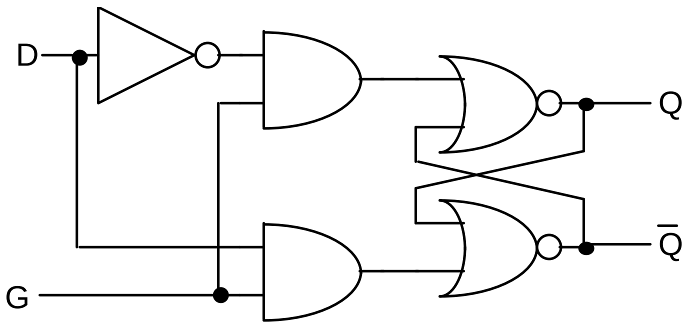


Figure 8: Latch schematic using ordinary gates

Because we have the freedom to work at the transistor level and build circuits like tristate inverters, we can greatly reduce the transistor count. A better latch uses two tristates and an inverter, requiring only 10 transistors.

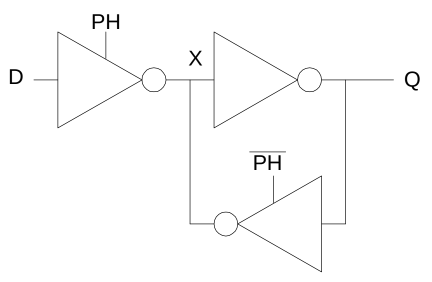


Figure 9: Improved latch schematic using tristates

When the clock PH is high, the input tristate is ON and the feedback tristate is OFF, so D flows through X to Q and there is no contention from the feedback. When PH falls, the input tristate turns off, which might leave node X floating. However, the feedback tristate turns on, recycling the value from Q back to X to hold a stable output value.

3. Schematic Entry & Simulation with Electric

This tutorial will show you how to design VLSI chips using a computer-aided design tool called Electric. Electric is the open-source brainchild of a brilliant programmer named Steve Rubin. It is still in the process of development and has its quirks and instabilities. However, virtually all complex CAD packages have quirks. Several of the advantages of Electric are its cross-platform support (Windows, Unix, and Mac), its price (free), access to the source code to enhance the features, and Steve's amazing turnaround fixing bugs and improving the tool. Disadvantages include its tendency to crash and its weaknesses importing designs from other CAD programs.

One of the major goals in this class is to find more problems with Electric and improve the tool until it is a stable and productive design environment. You will certainly run into frustrations along the way; this is typical of almost any cutting-edge field in which the tools have not caught up with design practices. You can help by submitting detailed reports of the problems you encounter so that Dr. Rubin can isolate the problem. You will receive extra credit for your reports if they are reproducible.

To submit a bug report, email Prof. Harris at David_Harris@hmc.edu with the following information:

Your name:
Date:
Facet:
A facet demonstrating the problem: list the library name, facet name, and facet view and attach the library to the email.
Description:
A detailed description of the problem, including the exact steps necessary to reproduce the problem.

Thank you for your patience with the tool. Your work submitting bug reports will make the tool better for the entire design community!

In this section, you will learn to draw schematics using transistors and build more complex schematics out of icons. You will also learn to simulate your schematics to check that they perform as you expected.

3.1. Getting Started

The latest version of the Electric CAD tools is kept on Kato at Harvey Mudd at home\Eng\Classes\E158\Electric. Check the date to be sure you run the latest version; the tool is updated often to fix bugs and add features. You may work from any Windows NT machine. Your personal PC or the Engineering Design Center computers are good choices. Electric is also available for the Mac and Unix. However, we will only be receiving frequent bug fixes for the NT version, so that is the recommended platform this semester. You may wish to map the Eng\Class directory as a network drive for convenient access. If you are not at Harvey Mudd, you can download Electric for free from http://www.staticfreesoft.com/. Note that you will need the Microsoft Visual C compiler to compile the source code.

Double-click on Electric to start the program. Dismiss the splash screen. Choose Info ➢ See Manual from the menu to bring up the Electric manual in a web browser. If the manual is not installed in the Electric folder, you can also find it at the Static Free Software web page as http://www.staticfreesoft.com/manual/. Skim through the following sections:

Chapter I: 1-2, 6-9
Chapter II: 1-6
Chapter III: 1-12

Don't worry if it doesn't all make sense yet. After you complete this lab, go back and skim over the sections that you initially found confusing. Refer back to the other chapters of the manual as you need help with specific features of Electric.

3.2. Schematic Entry

Your first exercise is to create a schematic for a 2-input NAND gate. Recall that each design is kept in a *facet*; for example, your schematic will be in the nand2{sch} facet, while your layout will eventually go in the nand2{lay} facet and your AND gate will go in the and2{sch} facet. Choose File • New Library and create your lab1_xx library where xx are your initials. Choose Facet • Edit Facet to bring up the Facet dialog. Click New Facet. Enter nand2 as the facet name and schematic as the view. A new editing window will appear with the title lab1_xx:nand2{sch} indicating the library, facet name, and view.

Electric defines various technologies for schematics and layout. To draw transistor-level schematics, you will need to select the Analog Schematic technology by choosing Technology • Change Current Technology and selecting the *schematic, analog* technology. This technology file contains basic circuit elements such as transistors, resistors, capacitors, and power and ground.

Your goal is to draw a gate like the one shown in Figure 10. Choose Windows • Toggle Grid to turn on a grid to help you align objects. Left-click on an NMOS transistor symbol in the components menu on the left side of the screen. Left-click on your layout to drop the transistor into your layout. Repeat until you have two NMOS transistors, two PMOS transistors, the circular power symbol, and the triangular ground symbol arranged on the page. You may move the objects around by left-clicking and dragging. The transistors default to a width/length value of 2/2. We will ignore this for now, but will need appropriate sizes in layout.

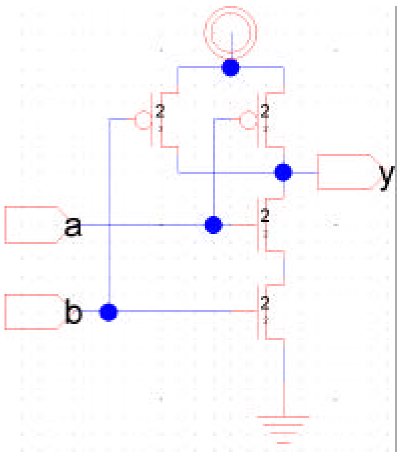


Figure 10: nand2{sch}

Now, make the connections. Left-click on a port such as the gate, source, or drain of a transistor. Right-click on another port to create a wire connecting the ports. You may also right-click in space to create a wire ending in space, then continue right clicking elsewhere to introduce bends. Continue until all the blue wiring to transistors and power and ground is completed.

Finally, you will need to provide *exports* defining inputs and outputs of the facet. Click on the pin symbol (the rectangle with a pointy end) in the components menu and drop it on the page. A dialog will pop up asking to give the export a name and a direction; make it an input named *a*. Repeat with input *b* and output *y*. Drag the pins to a place near the wires you have already drawn, then left and right click to connect wires to the pins. Remember that you must explicitly draw an arc to connect two nodes; it is easy to get in trouble by placing two objects so that they touch and appear to be connected but are not actually linked.

Use File • Save to save your library. Get into the habit of saving often because Electric crashes fairly often. Also, learn the keyboard shortcuts for the commands you use frequently.

3.3. Switch-Level Simulation

Our next step is to simulate the schematic to ensure it is correct. Electric has a built-in *switch-level simulator* called IRSIM. Such a simulator treats transistors as switches that may be ON or OFF. IRSIM understands something about the resistances and capacitances of transistors so that it can estimate delay.

Note that at the time of this writing, the IRSIM interface is complete. These instructions refer to the older built-in ALS simulator instead.

Start the simulation by selecting Tools • Simulation • Simulate. A waveform window will appear listing each of the *nets* in your design. If you created your exports properly, you will see nets for A, B, and Y, as shown in Figure 11. You will also see unnamed nets for VDD, GND, and the node between the series NMOS transistors. In this simulation, we can tell NET1 is power (VDD) and NET4 is ground because they are strongly driven high and low as indicated by the black lines. Thus, NET3 must be the node between the two NMOS transistors. The time scale at the bottom of the simulator is arbitrary and bears no relation to actual circuit performance.

The simulator has two vertical red cursors. The primary cursor with no x is used to create stimulus. Click and drag the cursor to about 40 ns. Click on the A input in the simulation window and press *h* or 1 to drive the input high. Drag the cursor to some later time. Click on the B input and press h to drive it high. Use the *l* or 0 key sometime later to drive A and B back low, as shown in the figure. Check that the Y output matches the behavior you would expect for a NAND2 gate. If it does not, fix the bug in your schematic and resimulate. The secondary cursor with an x is only used to measure delays relative to the first cursor. You will have no reason to use it because the delays are arbitrary anyway.

If you position the simulation and schematic window so that you can see both simultaneously, you can watch the color coding of the wires in the schematic change as you drag the primary cursor back and forth. These colors correspond to the voltage levels on the wires, with blue indicating low and magenta high. Beware that this is not consistent with the color coding of the waveform window! Watching the voltage levels change on the schematic is helpful for debugging problems. Study your simulation and determine why NET3 behaves as it does.

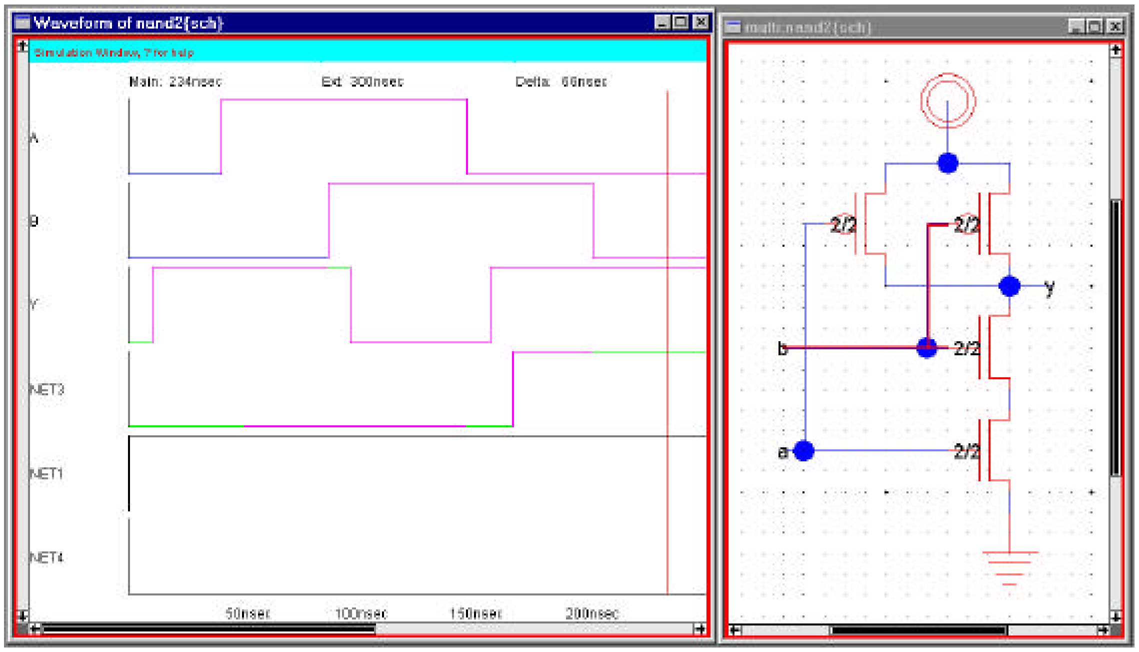


Figure 11: Simulation of nand2{sch}

When you request a simulation, the schematic is translated into a VHDL netlist. VHDL stands for VHSIC Hardware Description Language, and VHSIC in turn was a Department of Defense project on Very High Speed Integrated Circuits. VHDL is similar to Verilog, though somewhat more verbose. The VHDL in turn is translated into an internal text netlist format called net-als. Use the Facet • Edit Facet command to view the nand2{vhdl} facet. You should see something like this:

```
-- VHDL automatically generated from facet nand2{sch}
entity nand2 is port(b, a: in BIT; y: out BIT);
  end nand2;
architecture nand2_BODY of nand2 is
  component PMOStran port(g: in BIT; s, d: inout BIT);
    end component;
  component ground port(gnd: out BIT);
    end component;
  component power port(pwr: out BIT);
    end component;
  component nMOStran port(g: in BIT; s, d: inout BIT);
    end component;
  signal net3, net4, net1: BIT;
begin
  node7_1_1: PMOStran port map(b, y, net1);
  node3: ground port map(net4);
  node5: power port map(net1);
  node7: PMOStran port map(a, y, net1);
  node8: nMOStran port map(b, net3, y);
  node9: nMOStran port map(a, net4, net3);
end nand2_BODY;
```

A VHDL file is divided into entity and architecture descriptions of each cell. The opening *entity* line defines the inputs and outputs. The *architecture* describes the components used by the NAND gate (NMOS and PMOS transistors and power and ground) and how these components are connected. Study the file and see how it relates to the schematic you have drawn. Do the same for the net-als file. If you have simulation problems, it is sometimes helpful to examine the vhdl or net-als files to ensure that the input to the simulator matches what you'd expect.

Get in the habit of simulating each facet after you draw it so you catch errors while the design is fresh in your mind.

3.4. Inverter

Create a new schematic facet named inv. Draw a schematic for an inverter. Name the input *a* and the output *y*. Simulate the inverter to verify that it is correct.

3.5. Hierarchical Design

Now that you have a 2-input NAND gate and an inverter, you can use these cells as building blocks to construct a 2-input AND gate. Such hierarchical design is very important in the design of complex systems. You will find that the layout of an individual cell can be quite time consuming. It is very helpful to reuse cells wherever possible to avoid unnecessary drawing. Moreover, hierarchical design makes fixing errors much easier. For example, if you had a chip with a thousand nand gates and made an error in the nand design, you would prefer to only have to fix one nand cell so that all thousand instances of the nand inherit the correction than to need to fix each nand individually.

Each schematic has a corresponding symbol, called an *icon*, used to represent the cell in a higher-level schematic. \You will need to create an icon for your 2-input NAND gate and your inverter.

Open your nand2{sch} and choose View • Make Icon. Electric will create a generic icon based on the exports looking something like Figure 12. It will drop the icon in the schematic for handy reference; drag the icon away from the transistors so it leaves the schematic readable.

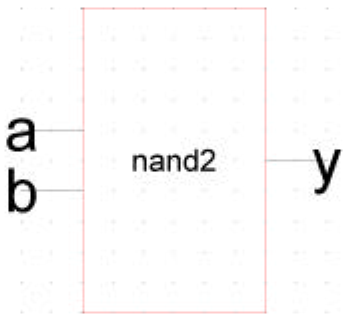


Figure 12: nand2{ic} from Make Icon

A schematic is easier to read when familiar icons are used instead of generic boxes. You will modify the icon to look like Figure 13. Pay attention to the dimensions of the icon; the overall design will look more readable if icons are of consistent sizes.

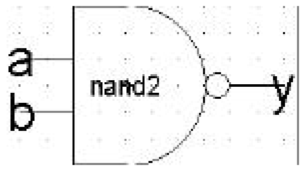


Figure 13: nand2{ic} final version

Click on the icon and choose Facets • Down Hierarchy to drop in to the icon. The technology will automatically change to *artwork*. A palette will appear with various shapes. Delete the generic red box but leave the input and output lines.

Place a *facet center* in the middle[2] of the icon using the Edit • New Special Object • Facet Center command. Drag the facet center where you want it. If you deselect it, then try to select it again, you'll find you can't pick it up. The facet center is by *default hard to select*. To select hard to select nodes on a Windows machine, hold the Alt key down while clicking. On other machines, see the Special Select key combination in section 1.8 of the Electric manual. Using facet centers on all your icons and layout facets will avoid nasty off-grid errors later in your work.

The body of the NAND is formed from an open C-shaped polygon, a semicircle, and a small circle. To form the semicircle, place an unfilled circle. Double-click to change its size to 6x6 and to span only 180 degrees of the circle. Use the rotate commands under the Edit menu to rotate the semicircle into place. Place another circle and adjust its size to 1x1. You may change the alignment options under the Windows menu to 0.5 to move the circle into place, then set alignment back to 1. Equivalently, you may press the h key on the keyboard to indicate movement by half units. Select the small circle and use the arrow keys on the keyboard to move it into place. Then press the f key on the keyboard to restore arrow key movement to full units.

The opened-polygon shown in Figure 14 can be used to form the C-shaped body. Drop an opened-polygon object. Select it and choose Edit • Special Function • Outline Edit to enter outline edit mode. In this mode, you can use the left button to select and move points and the right button to create points. You should be able to form the shape with four clicks of the right button to define the four vertices. Outline edit mode is not entirely intuitive at first, but you will master it with practice. Choose Edit • Special Function • Exit Outline Edit when you are done. If your shape is incorrect, delete it, drop another opened-polygon, and try again.

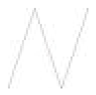


Figure 14: Opened-Polygon

Electric is finicky about moving the lines with inputs or outputs. If you left-click and drag to select the line along with the input, everything moves as expected. If you try to move only the export name, it won't move as you might expect. Therefore, make a habit of moving both the line and export simultaneously when editing icons. Note that the line is just an open-polygon and can be shortened if desired by entering Outline Edit mode.

Finally, place the name on the icon by choosing Edit • New Special Object • Text (nonlayout). Click in the center of the icon and type *nand2*.

When you are done, use Edit • Up Hierarchy to return to the schematic. You may find it helpful to use the fill window and zoom commands under the Windows menu to zoom in

[2] It is not strictly necessary for the facet center to be in the true center of the facet. Electric just uses the center to provide a reference point within the facet. This helps Electric keep cells on a clean grid.

and out of your design. Then open the inverter schematic and create an icon for it as shown in Figure 15.

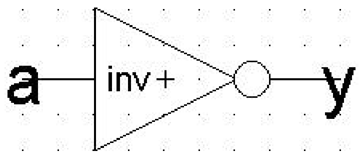


Figure 15: inv{ic}

Now that you have icons for the nand2 and inverter, create a new schematic called and2. Change the technology back to *schematic, analog* because you are drawing a schematic again now. Use Edit • New Facet Instance to create ("*instantiate*") the nand2{ic} and an inv{ic}. Wire the two together and create exports on inputs *a* and *b* and output *y*. Double-click on the wire between the two gates and give it a name like *yb* so you know what you are looking at in simulation. When you are done, your and2 schematic should look like Figure 16. If the line between the gates is black rather than blue, you neglected to return to the *schematic, analog* technology and were still drawing using the *artwork* palette.

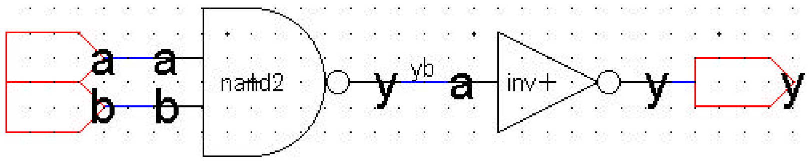


Figure 16: and2{sch}

Simulate your and2 gate to ensure it works. Try the Tools • Simulation • Down Hierarchy command to descend into each of the gates and observe their internal waveforms. The gates are given generic names such as NODE4 and NODE5 when created, so it can be hard to determine which is the nand and which is the inverter. In the schematic editor, you can double-click on each gate and assign it a name to help differentiate cells when you simulate.

3.6. Other Gates

Draw schematics for the following gates. Simulate that they work correctly. When you draw the latch, use two tristates and an inv. Make ph and phb of type clock rather than input.

Cell	Inputs	Outputs
nor2	a, b	y
or2	a, b	y
tristate	a, en, enb	y
latch	d, ph, phb	q

4. Transistor Fabrication

We can build useful digital circuits from logic gates and logic gates from transistors. Our final step to understand chip design is to explore how transistors are built. We'll begin by examining the physical structure of CMOS transistors. Then we'll look at a manufacturing process to build these structures. Finally, we'll see what "masks" you must specify to manufacture transistors and wires in the locations you would like.

4.1. Semiconductor Physics

CMOS transistors are constructed from silicon. Silicon is a *semiconductor*, meaning it is neither a good insulator like your sandals nor a good conductor like Michael Tilson Thomas. Silicon is a group IV material so it has four electrons in its valence shell and forms four bonds to adjacent silicon atoms in a crystal lattice.

When impurities, called *dopants*, are introduced into silicon, the electrical properties change. If some of the silicon atoms are replaced with a group V material like arsenic, there will be an extra electron after the four bonds are satisfied. This electron is not part of a bond, so it is free to wander about the lattice. Thus, the electrical conductivity becomes much better. We call silicon doped with a group V material an n-type semiconductor because it conducts using negative carriers.

Similarly, if a group III dopant such as boron is introduced, some atoms in the lattice will be missing a bond. This missing bond may be filled with an electron from a neighboring atom, which now in turn will be missing an electron. These missing electrons are called *holes* and can propagate through the lattice just like excess electrons, but effectively act as a positive charge. We call silicon doped with a group III material a p-type semiconductor.

n-type semiconductors and p-type semiconductors can be reasonably good conductors, especially if heavily doped. However, current does not flow from n-type to p-type semiconductors.

4.2. CMOS Transistor Operation

A cross-section of an NMOS transistor is shown below. Two regions of heavily doped n-type silicon (n+) form the source and drain. They are separated by a p-type substrate. A conducting gate is separated from the substrate by a thin layer of insulating SiO_2[3]. Once upon a time, the gate was made from metal. Thus, MOS transistors got their name from the Metal-Oxide-Semiconductor stack formed by the gate, insulator, and substrate. Since the late 70's, the gate has actually been made from heavily doped silicon rather than metal because the manufacturing is easier, but the name MOS stuck.

[3] SiO2, silicon dioxide, is a fancy name for glass. Chip designers affectionately just call it "*oxide*."

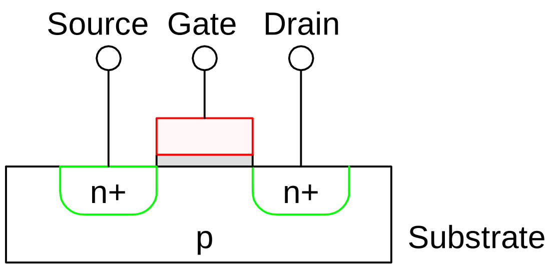


Figure 17: NMOS transistor cross-section

When the gate voltage is low, the substrate isolates the source and drain, so no current flows between the two. We say the transistor is OFF. Notice that the gate and substrate act as two plates of a capacitor separated by a very thin oxide dielectric. When the gate voltage becomes high relative to the substrate and source, we expect positive carriers will be attracted to the positive terminal of the capacitor (the gate), and negative carriers will be attracted to the negative terminal of the capacitor (the substrate). When the gate voltage exceeds a certain threshold, typically 0.35-1 volt, enough negative carriers accumulate just below the oxide layer to "invert" the substrate and form a thin channel of n-type semiconductor. Now we have an n-type conducting path between the source and drain, so current can flow between those terminals. We say the transistor is ON.

PMOS transistors work in much the same way, but switch the dopings. The source and substrate are normally held at a high voltage. When the gate is at a high voltage too, gate capacitor has no voltage across it and the transistor is OFF. When the gate is at a low voltage, positive carriers are attracted from the substrate to just below the oxide and the transistor turns ON.

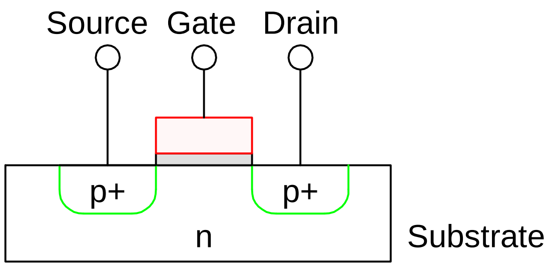


Figure 18: PMOS transistor cross-section

Notice that the substrate is effectively a fourth terminal of the transistor that we have not discussed earlier. This fourth terminal is sometimes referred to as the "bulk." It should be at GND for an NMOS transistor and at VDD for a PMOS transistor to make the gate operate correctly.

Transistor performance is determined by the transistor length and width. The length is the distance from source to drain. Shorter channel lengths offer faster transistors because the current has less distance to flow, so digital circuits generally use the shortest channels that can be reliably manufactured. The width is the distance into the page in the figures above. Wider transistors provide more current, but also have more capacitance so the best choice of width depends on the application.

4.3. Inverter Cross-Section

An inverter requires both an NMOS and a PMOS transistor, connected together and connected to VDD and GND. The NMOS transistor wants a p-type substrate, while the PMOS transistor wants an n-type substrate. Transistors are built on a single substrate, so somehow we must be able to locally change the doping to accommodate both flavors of transistors. A common approach is to use a thin wafer of n-type silicon as the substrate with "wells" of p-type silicon. The illustration below shows a cross-section of an inverter.

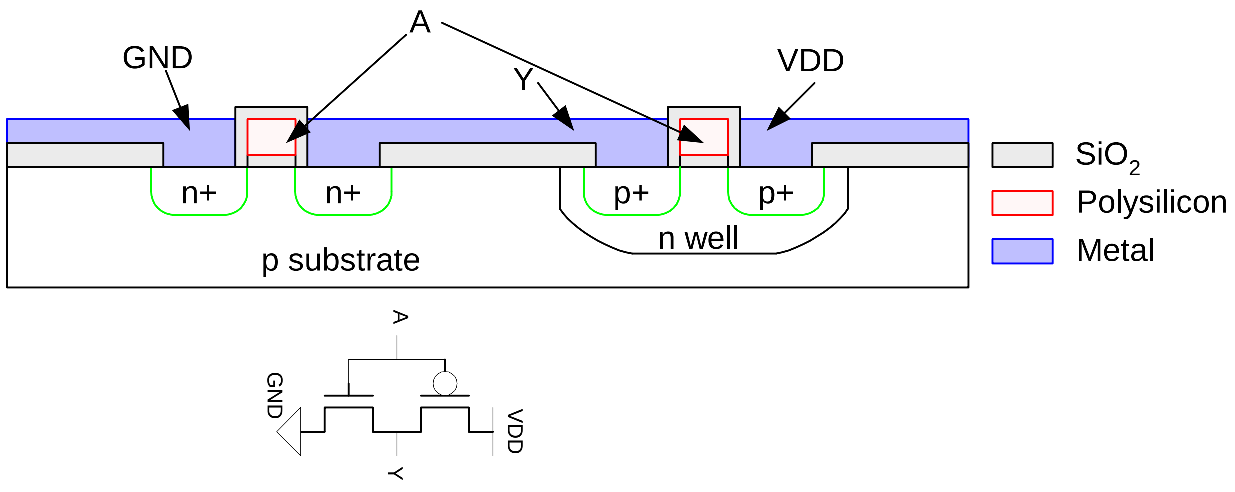


Figure 19: Inverter cross-section

The surface of the chip is covered with an insulating layer of oxide to prevent undesired short connections. Then contact cuts are made and metal is deposited. The metal connects the source of the PMOS transistor to VDD and the source of the NMOS transistor to GND. The two drains are tied together to the output Y. The gates are formed from polycrystalline silicon (called polysilicon for short, or just poly if you are feeling especially terse) and are connected outside of the cross-section shown in this illustration.

We had stated earlier that the p-type substrate should be at a low voltage and the n-well should be at a high voltage for correct transistor operation. We can add heavily doped substrate and well taps to take care of this, as shown below. The p+ substrate contact allows a connection from GND to the substrate. The n+ well contact allows a connection from VDD to the well. Using heavily doped silicon contacts forms a better connection than a simple metal to lightly-doped silicon junction.

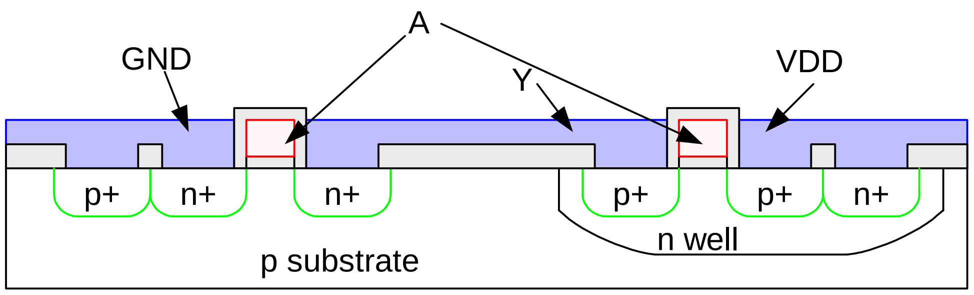


Figure 20: Inverter cross-section with well and substrate contacts

4.4. Fabrication Sequence

Chips are amazingly inexpensive for all their complexity because all the transistors and wires can be printed in much the same way as books are printed. The fabrication sequence consists of a series of steps in which layers of the chip are defined through a process called photolithography. Because many entire chips are printed at once, the cost of the chip is proportional to the chip area, rather than the number of transistors. As manufacturing advances allow engineers to build smaller transistors and place more transistors in the same area, each transistor gets cheaper. Smaller transistors are also faster because electrons don't have to travel as far to get from the source to the drain! This explains the remarkable trend for computers and electronics to become both cheaper and more capable with each generation.

Let us trace a simple sequence of steps to fabricate our inverter. We will use a set of six masks to define the parts of the inverter: n-well, polysilicon, n+ diffusion, p+ diffusion, contacts, and metal. Masks define where the parts will be manufactured on the chip. The masks for the inverter are shown stacked up in a top view. The cross-section of the inverter corresponds to the dashed line.

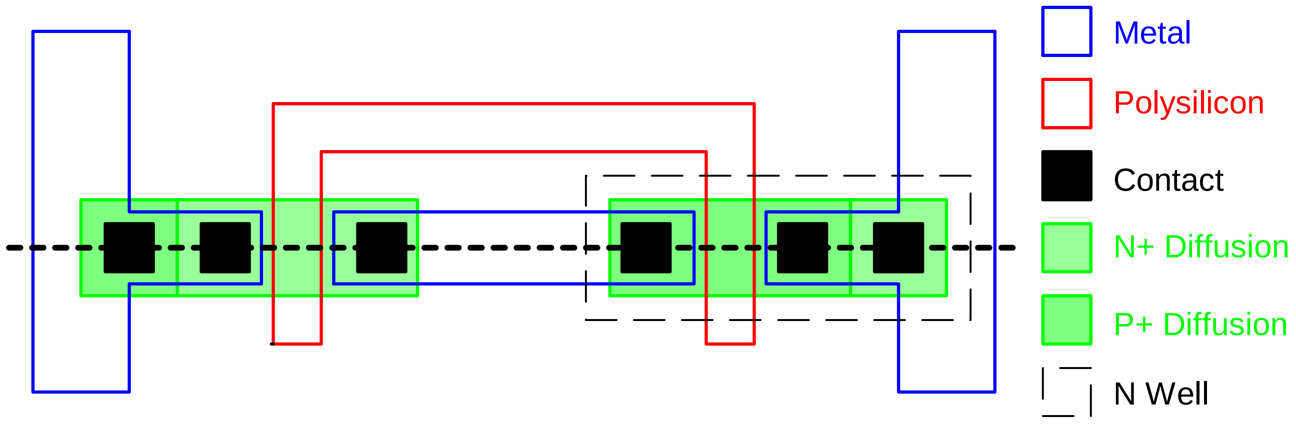
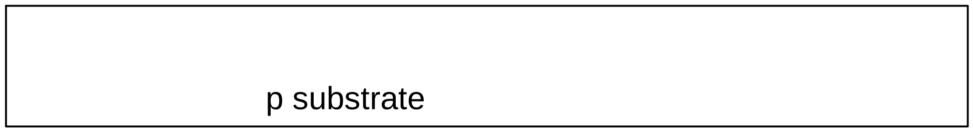


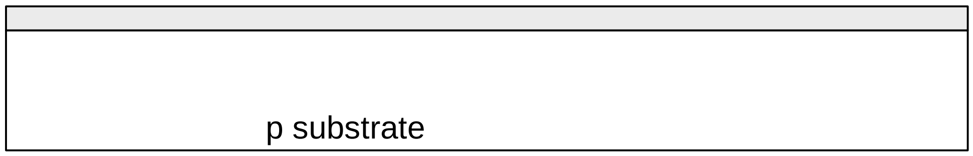
Figure 21: Inverter mask set

We begin with a bare p-type silicon wafer that serves as our substrate. We'll use a very simple fabrication process to illustrate the fundamental ideas; practical fabrication processes extend this general approach to build circuits with better performance.

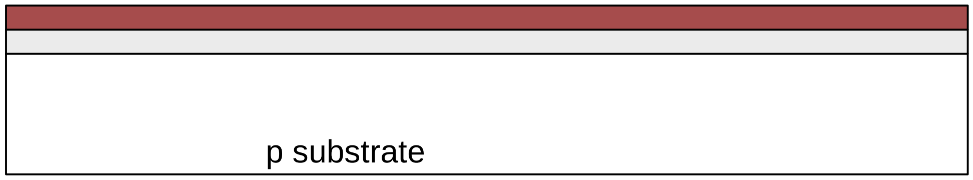


We first form the n-well. This requires adding enough group V dopants into the silicon substrate to change the substrate from p to n-type in the region of the well. To define what regions receive n-wells, we grow a layer of oxide over the entire wafer, then remove it where we want the wells. We then add the n-type dopants; the dopants are blocked by the oxide, but enter the substrate and form the wells where there is no oxide.

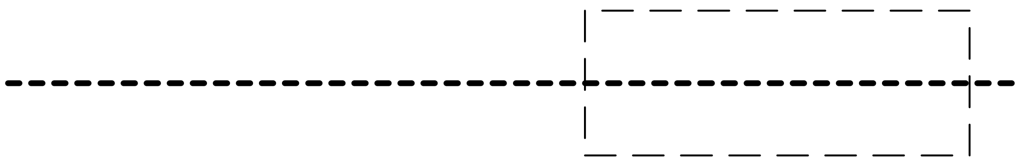
Let us elaborate, looking at a cross-section of the wafer as we proceed. We grow the oxide layer by exposing the silicon wafer to oxygen in a high-temperature furnace that causes the Si and O_2 to react and become $Si0_2$ on the wafer surface.



We now need to pattern the wafer to define the n-well. We spin on an organic photoresist[4] that softens where exposed to light.

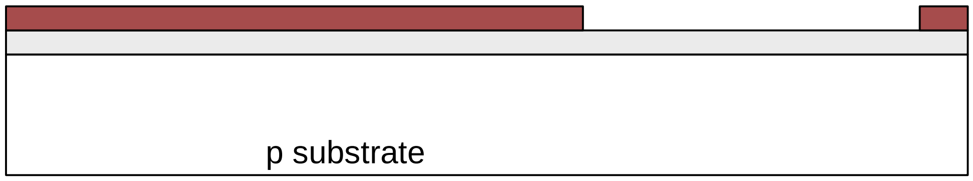


We expose the photoresist through the n-well mask that allows light to pass through only where we want the well. We then remove the remaining photoresist. Figure 22 shows the top view of the N-well mask looking down at a wafer. The cross-section below shows where the photoresist is removed to prepare to form the well.

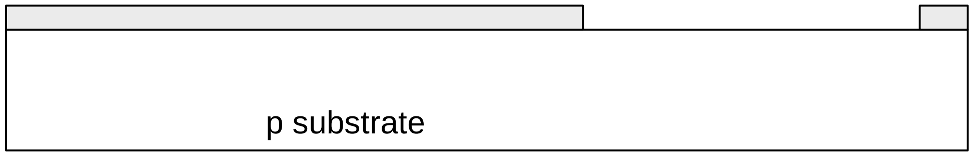


[4] Engineers have experimented with many types of compounds for photoresist. Brumford & Walker reported in 1958 that Jello™ could be used for masking. They did extensive testing, noting that "various Jellos™ were evaluated with lemon giving the best result."

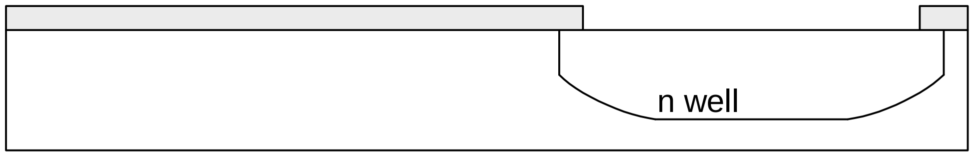
Figure 22: n-well mask



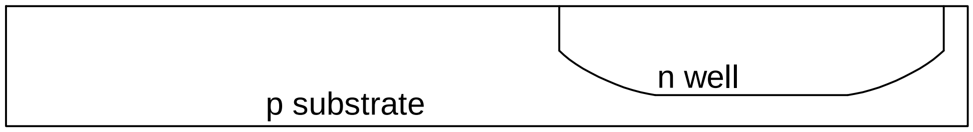
Now we etch the oxide using HF, hydrofluoric acid[5], where it is not protected by the photoresist. Finally we strip away the hardened photoresist using a mixture of acids called piranha etch.



Now we form the wells where the substrate is not covered with oxide. Two ways to add dopants are diffusion and ion implantation. With diffusion, we place the wafer in a furnace with a gas containing the dopants. When heated, dopant atoms diffuse into the substrate. Notice how the well is larger than the hole in the oxide. Although most diffusion goes downward into the substrate, some lateral diffusion occurs sideways. With ion implantation, we accelerate dopant ions through an electric field to blast them into the substrate. In either method, the oxide layer prevents dopant atoms from entering the substrate where no well is intended.

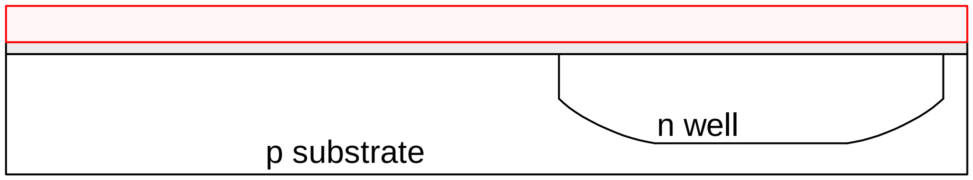


Finally, we strip the oxide using HF again to return to a bare wafer with wells in the appropriate places.



Next we will form the gates of the transistors. These consist of polysilicon over a thin layer of oxide. Therefore, we grow the thin oxide in a furnace again. Then we place the wafer in another reactor with silane gas and heat again to grow the polysilicon layer through the process called chemical vapor deposition.

[5] HF is particularly vicious because it painlessly seeps through skin to eat the bone beneath. Proper safety procedures are vital in the fab.



We again apply photoresist and pattern it to define the polysilicon. Each step of the fabrication process will use photoresist and an oxide for patterning just as was done for n-wells, so we will not show it from here on. We are left with the polysilicon gates.

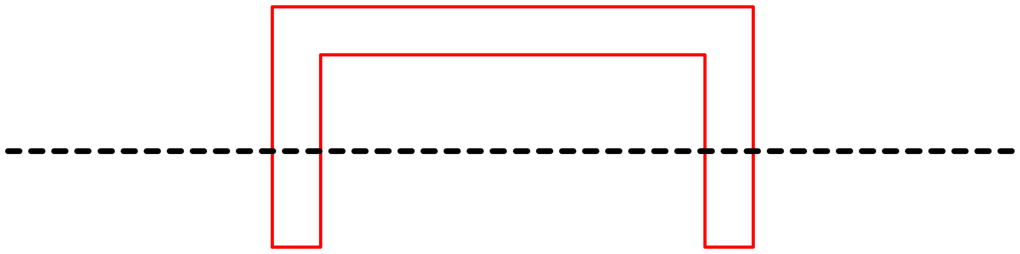
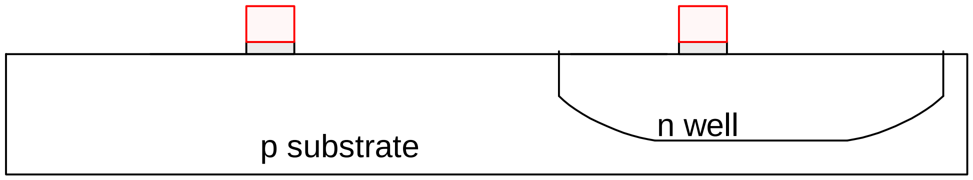
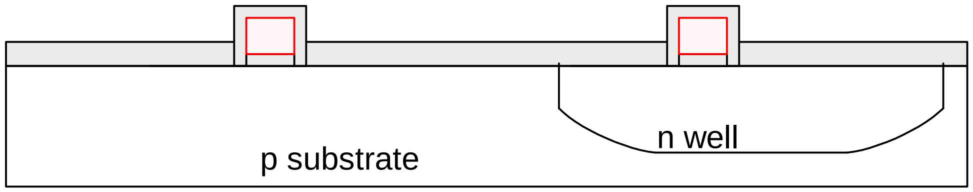


Figure 23: Polysilicon mask



We next form the n+ diffusion. As usual, we grow an oxide layer covering the wafer.



We use the n+ diffusion mask to define where the NMOS transistors and n-well contacts go.

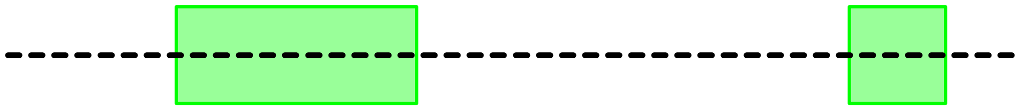
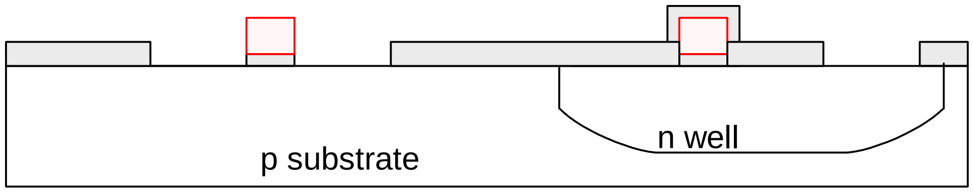
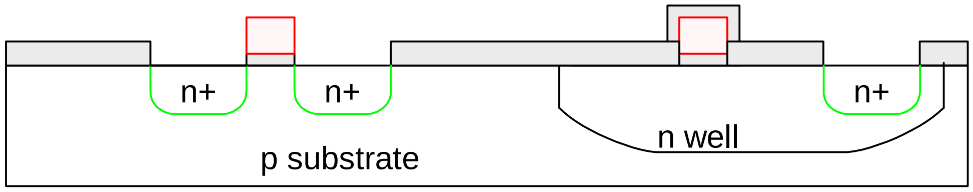


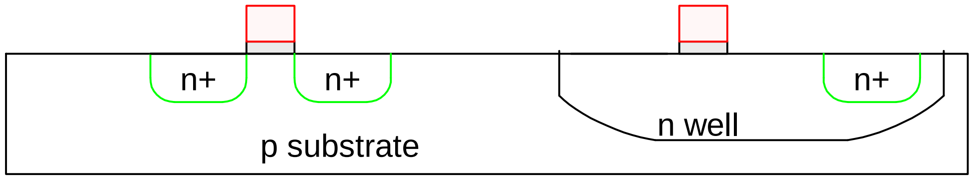
Figure 24: n+ Diffusion mask



We use diffusion (or now often ion implantation) to form the n+ regions where the oxide is missing. Notice that the polysilicon gate over the NMOS transistor blocks the diffusion. This is called a self-aligned process because the source and drain of the transistor are automatically formed adjacent to the gate without the need to precisely align the masks.



Again we strip off the oxide to leave the bare wafer.



We repeat the process for p+ diffusion. Oxide is used for masking in the same way, and thus is not shown.

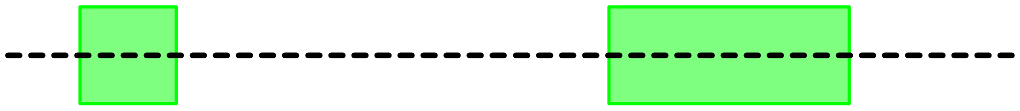
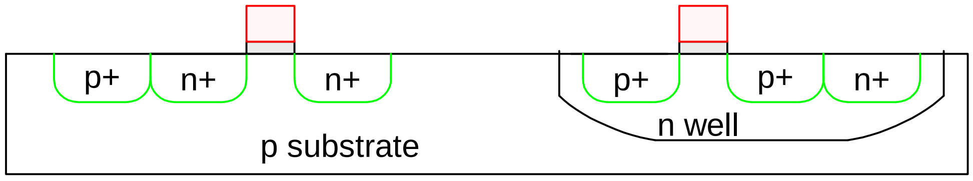


Figure 25: P+ Diffusion mask



We cover the transistors with a thick oxide layer to insulate them from the metal layer. Where we want to connect metal to the transistors, we must etch contact cuts defined by the contact mask.

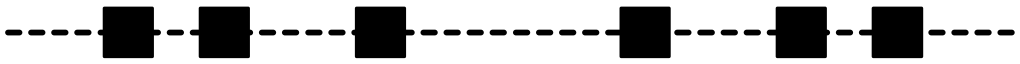
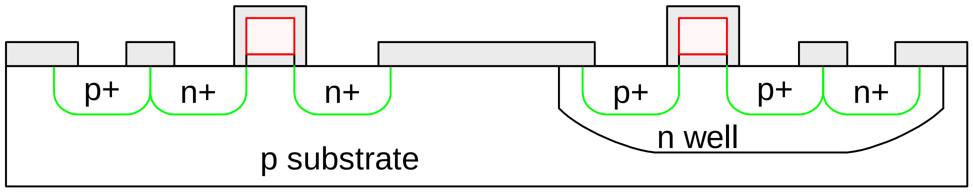


Figure 26: Contact mask



Finally, we sputter metal over the entire wafer, filling the contact cuts as well. Sputtering involves blasting aluminum into a vapor that evenly coats the wafer. We use the metal mask and a plasma etch to remove metal except where we want the wires. This completes our simple fabrication process.

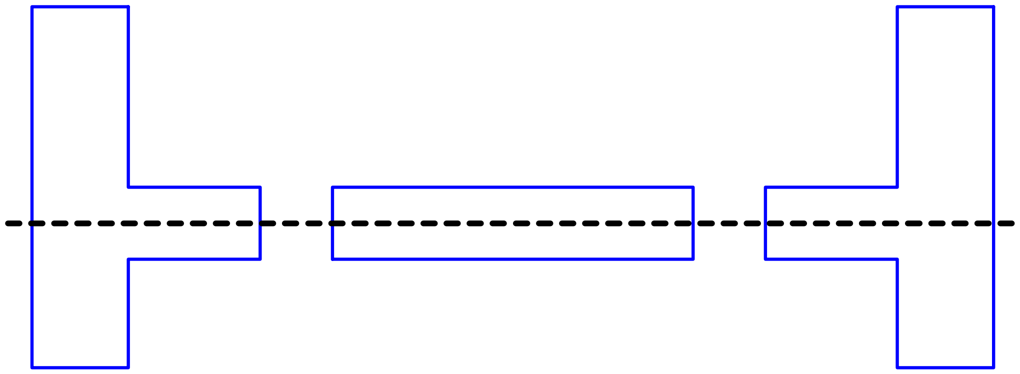
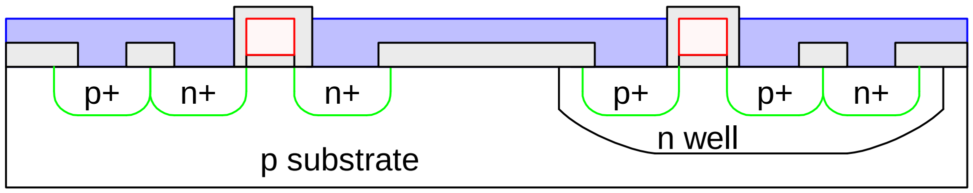


Figure 27: Metal mask



Modern fabrication sequences are somewhat more elaborate because they must create complex doping profiles around the channel of the transistor or to print features that are smaller than the wavelength of the light being used in lithography. However masks for these elaborations can be automatically generated from the simple set of masks we have just examined. Modern processes may also have five or more layers of metal, so the metal and contact steps must be repeated for each layer. Chip manufacturing has become a commodity and designers can contract to many different vendors to build designs given a basic set of masks.

4.5. Layout Design Rules

Layout design rules define the minimum dimensions of manufacturable features on the chip. It is convenient to express rules in terms of a parameter ? which is characteristic of the resolution of the process. ? is half the minimum length of a transistor gate. If we develop a set of layout design rules based on ?, we can take a design originally intended for an older process and scale it to a newer process by only changing ? rather than redrawing the layout.[6]

An easy way to remember design rules is that most materials must be 4 ? wide, but polysilicon is only 2 ?. Contacts are always 2x2. Spacing between materials is the same as material width. These rules are mostly conservative; many layers can be placed on a 3 ? width or spacing. 4 ? is convenient, though, because it keeps both the centers and edges of materials on integral multiples of ?.

The definitive design rules are available on the MOSIS web page. MOSIS is a service that collects small orders for designs and combines them into an order large enough to interest a semiconductor manufacturing facility. MOSIS has developed a set of *Scalable CMOS* design rules that are sufficiently conservative to work for virtually all processes. The original rules, *SCMOS*, apply to older processes. We will be using the *SUBM* submicron rules that are even more conservative and suffice for more advanced processes. They are adequate for both the AMI 0.5 and 1.5 micron processes. Finally, the *DEEP* deep submicron rules are slightly more conservative and are necessary for best performance in the 0.25 ?m and below processes. Tables 3 and 4 on the MOSIS web page list the differences between these design rules.

http://www.mosis.org/Technical/Designrules/scmos/scmos-main.html

We will be fabricating using the SCN3ME _SUBM technology code defined in Table 5 of the web page. SCNE means Scalable CMOS with n-wells and an Electrode layer, i.e. polysilcon 2. Click on each of the layers in the table to see the design rules. For example, the Metal1 rules are shown in Figure 28 below. We will follow the SUBM rule set, requiring metal width and spacing of 3?. Nevertheless, unless you are severely pressed for space, using a width and spacing of 4? will usually make your layout easier. Also note that the minimum spacing between lines increases for very wide metal lines like those you might use to carry power or ground.

[6] Not all design rules scale exactly linearly from one process to another, so ?-based design rules are necessarily somewhat conservative. Industry generally is willing to put more effort into porting a chip from one process to the next, so most industrial rules are expressed in microns to take maximum advantage of the process.

SCMOS Layout Rules - Metal1

Rule	Description	Lambda		
		SCMOS	SUBM	DEEP
7.1	Minimum width	3	3	3
7.2	Minimum spacing	2	3	3
7.3	Minimum overlap of any contact	1	1	1
7.4	Minimum spacing when either metal line is wider than 10 lambda	4	6	6

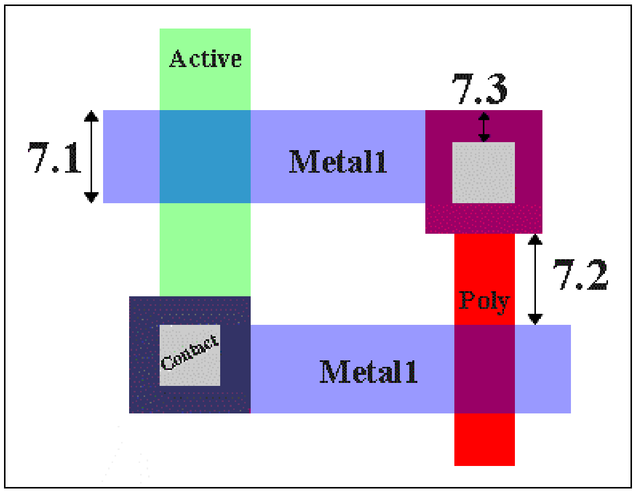


Figure 28: SCMOS Metal1 Rules

4.6. Standard Cell Layout Style

It is important to choose a consistent layout style for your cells so that you can “snap together” various different cells and easily wire them together.

In a process with one polysilicon layer and two metal layers, a convenient standard cell layout style is shown in Figure 29 for an inverter. Power and ground are run horizontally at the top and bottom of the cell in metal1; a consistent distance between power and ground (60 ? center to center in this example) is used for all cells to facilitate snapping the cells together. Inputs come from the top in polysilicon and outputs are routed to the top in metal2. Cells may be connected together using horizontal metal1 wires. Note that even though our initial example of inverter masks used a bent gate to illustrate a cross-section through both transistors, the standard cell style shown here is more conventional.

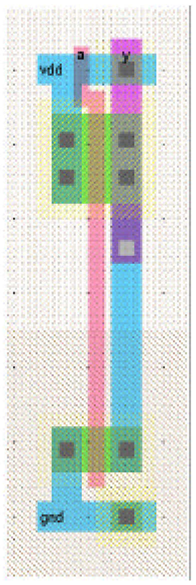


Figure 29: Inverter Layout

A few guidelines will give you better layout results. Use metal for lengthy wires; polysilicon and particularly diffusion are much more resistive and make slow wires. Place well and substrate contacts in each cell to ensure the well and substrate remain at VDD and GND. If the cells are especially large, place multiple contacts to ensure every transistor is within 100 ? of the appropriate contact. Devote the top part of the cell to PMOS transistors with n-well and the bottom part to NMOS transistors so that wells abut cleanly when cells are snapped together.

5. Layout Entry and Verification

Now that you have a schematic simulating correctly and understand how transistors are created, it is time to draw the layout. You will first layout a 2-input NAND gate and an inverter. You will join the two cells to form a 2-input AND gate. You will also learn to check that your design satisfies the layout design rules, has sufficient well contacts, matches the topology of the schematic, and simulates correctly.

5.1. Layout

Let us begin with the layout for a 2-input NAND gate. Choose Facet • Edit Facet to bring up the Facet dialog. Click New Facet. Enter nand2 as the facet name and layout as the view. We will be targeting the AMI 0.5 ?m process but using MOSIS submicron scalable rules so we could easily adapt to a more advanced or a less expensive process. To setup the technology file, choose Technology • Change Current Technology and select *mocmos*, the MOSIS CMOS technology. Then choose Technology • Change Units and set Lambda for mocmos to 600 half-milimicrons, i.e. 0.3 ?m[7]. Use the Info • New Arc Options to set the default width for Metal-1 to 4 and for Metal-2 to 4 for convenience of layout. Finally, choose Technology • Technology Options and select 2 metal layers, submicron rules, alternate Active and Poly contact rules, and disallow stacked vias. The submicron rules are documented on the MOSIS web page. Alternate Active and Poly contact rules are an older, cleaner set of design rules that don't involve half-lambda dimensions. Disallow stacked vias will cause the design rule checker to forbid vias placed on top of contacts. The mocmos technology shows two polysilicon in the palette. Do not use the orange Polysilicon-2 layer; it is shown in the palette but is not available in the AMI process.

Your goal is to draw a layout like the one shown in Figure 30.

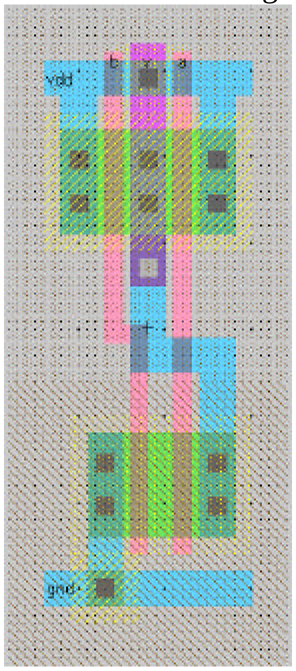


Figure 30: nand2{lay}

Start by drawing your NMOS transistors. Recall that an NMOS transistor is formed when polysilicon crosses n-diffusion. n-diffusion is represented in Electric as green diffusion surrounded by a dotted yellow n-select layer all within a hashed black P-well background. This set of layers is conveniently provided as a 3-terminal transistor node in Electric. Move the mouse to the components menu on the left side of the screen. As you move the mouse over various objects, the node name will appear on the status line next to the word NODE near the bottom left corner of the screen. Left click on the n-transistor, shown in Figure 31, and click again in the layout window to drop the transistor in place. Use the Edit • Rotate • 90 Degrees Counterclockwise command to rotate the transistor so that the red polysilicon gate is oriented vertically. There are two NMOS transistors in

[7] Although the process is called 0.5 ?m, l = 0.3 ?m means that transistor channel lengths are drawn at 2 ? = 0.6 ?m. Transistor lengths are automatically shrunk to 0.5 ?m for us when masks are generated.

series in a 2-input NAND gate, so we would like to make each wider to compensate. Double-click on the transistor. In the node information dialog, adjust the width to 12.

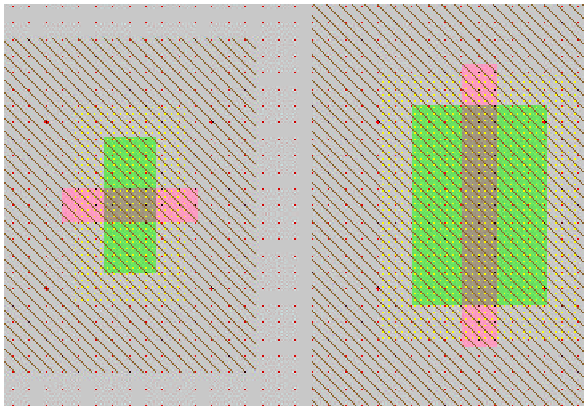


Figure 31: NMOS transistor before and after rotation and sizing

We need two transistors in series, so copy and paste the transistor you have drawn or use the Edit • Duplicate command. Drag the two transistors along side each other so they are not quite touching. Left click the diffusion (source/drain) of one of the transistors and right click on the diffusion of the other transistor to connect the two. Then drag the two transistors until the polysilicon gates are 3? apart, looking like they do in Figure 30. You will probably find it helpful to turn on the grid using the Windows • Toggle Grid command. The grid defaults to small dots every lambda and large dots every 10?. You can change this with the Windows • Grid Options command. Also by default, objects snap to a 1-??grid.

Electric has an interactive *design rule checker* (DRC). If you place elements too closely together, it will report errors in the Electric Messages window. Try dragging one of the transistors until its gate is only 2? from the other. Observe the DRC error. Then drag the transistors back to proper spacing. When you are in doubt about spacing, use the Tools • DRC • Hierarchical Check command to ask Electric to recheck the entire facet and any subfacets it might contain.

Next we will create the contacts from the n+ diffusion to metal1. Diffusion is also refered to as *active area*. Drop a square of Metal-1-N-Active-Contact in the layout window and double-click to change the properties to a Y size of 12. You will need a second contact for the other end of the series stack of NMOS transistors, so duplicate the contact you have drawn. Move the contacts near each end of the transistor stack and draw diffusion lines to connect the transistors. Then move the contacts even closer; you only need a gap of 1? between the metal and polysilicon. Use the design rule checker to ensure you are as close as possible but no closer. Using similar steps, draw two PMOS transistors in parallel and create contacts from the P-diffusion to metal1. At this point, your layout should look something like Figure 32:

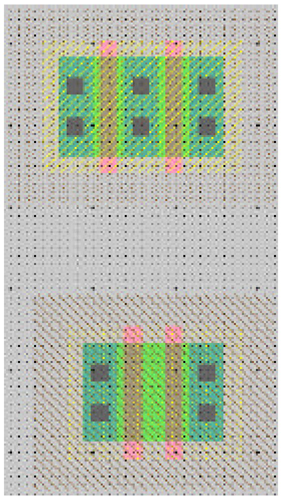


Figure 32: Contacted transistors for nand2 layout

Draw wires to connect the polysilicon gates, forming inputs *a* and *b*, and the metal1 output node *y*. Then add metal1 power and ground lines. You can add a line by placing a pin such as metal-1-pin from the palette then right-clicking nearby to draw the line. Use the grid to ensure power and ground are 60? apart from the center of each line. This is the same spacing as the power/ground lines of the inverter. Add more metal lines to connect power and ground to the transistors.

Recall that well contacts are required to keep the diodes between the wells and source/drain diffusion reverse biased. We will place an n-well contact and a p-well contact in each cell. Place the n-well contact under the ground rail and connect it to the ground via with metal1. Place the n-well contact under the power rail and connect it to VDD with metal1.

The inputs and outputs should arrive at the top of the cell in polysilicon and metal2, respectively. Extend the poly inputs to the top. Add a metal1-metal2 contact, also known as a via, to the output wire. Select the via, then click in the palette on the magenta metal2 arc box to tell Electric that you would like to draw your next wire from the via in metal2 rather than metal1. Right-click above the cell to extend a metal2 line from the via upward to form the output.

Electric is agnostic about the polarity of well and substrate; it generates both n- and p-well layers. In our process that has a p-substrate already, the p-well, indicated by black slanting lines, will be ignored. The n-well, indicated by small black dots, will define the well on the chip. Electric only generates enough well to surround the n and p diffusion regions of the chip. It is a good idea to create rectangles of well to entirely cover each cell so that when you abut multiple cells you don't end up with awkward gaps between wells that cause design rule errors. In many cells, the PMOS transistors will be larger than the NMOS transistors. Therefore, we will dedicate from 5? below the center of the design downward for n-wells containing PMOS transistors and from 5? below the center of the design upward for p-wells containing NMOS transistors.

To do this, choose Edit • New Pure Layer Node. Create an N-Well-Node and a P-Well-Node. Double-click on each to change the N-Well-Node to 32 wide by 43 tall so that it entirely covers the existing n-well and extends down to 5 lambda below the centerline. Similarly, change the P-Well-Node to 32 wide by 33 tall. You will find the pure layer nodes are annoying because you will tend to select them when you really wanted to select a transistor or wire. To avoid this problem, shift-click to select both pure layer well nodes and use the Edit • Selection • Make Selected Hard to make the layers hard to select. You will then need to hold the Alt key while clicking to select a well. You can use the Make Selected Easy command if you want to restore a well to be easily selected.

Place a facet center roughly in the center of the cell using the Edit • New Special Object • Facet Center command, just as you had with an icon. Remember that facet centers are also hard to select.

Finally, provide exports for the cell. When you use the cell in another design, the exports define the locations that you can connect to the cell. Click on top end of the metal2 output wire. You will see a small white box highlighted, corresponding to the pin at the end of the cell. If you accidentally selected the entire line instead, click elsewhere in space to deselect the line, then try again to find the pin. You may also try holding the *ctrl* key while clicking to cycle through selections. Add an output export called *y*. Repeat for input ports *a* and *b*. Also export *vdd* and *gnd* from the metal 2 lines; these should be of type power and ground, respectively. Electric recognizes vdd and gnd as special names, so be sure to use these names.

Your design should now resemble Figure 30 and should pass hierarchical DRC. Fix any DRC errors you might discover. Save the library.

Electric has a fun feature to show a 3D rendition of your layout. Use the Windows • 3D Display • View in 3 Dimensions command. Click and drag with the mouse to rotate the layout. You will see the layer stack from diffusion on the bottom through polysilicon, metal1 and metal2. Vias are shown in white and contacts from metal1 to poly or diffusion in black. Does the 3-D visualization match your mental picture of the layout? When you are done, use Windows • 2D Display • View in 2 Dimensions to restore normal viewing.

5.2. Layout Verification

Layout verification involves more checks than schematic verification. The checks include *Design Rule Checks* (DRC), *Electrical Rule Checks* (ERC), *Network Consistency Checking* (NCC), and simulation.

You will likely find yourself invoking these menu options often and may wish to give yourself keyboard shortcuts. For example, you might use the Info • User Interface • Quick Key Options dialog to map the Tools • DRC • Hierarchical Check command to the F1 function key, Tools • Network • Network Options to F2, and Tools • Electrical Rules • Analyze Wells to F3.

First, be sure the design satisfies the layout design rules by running a hierarchical DRC. This checks the layout and any facets it might incorporate; in this case the nand2 is a *leaf* cell and has no subfacets. You should have already corrected any design rule violations.

Next, run Tools • Electrical Rules • Analyze Wells. This check ensures that every n-well has a contact to VDD and every P-well has a contact to GND reasonably close by. ERC will report the number of transistors found Check that this matches your expectation; it should be 2 NMOS and 2 PMOS for the nand2. It also reports the farthest distance of any part of the layout from a well or substrate contact; if this is greater than 100, consider adding more contacts to avoid latchup risks. If any errors are reported, fix them.

Next, simulate the layout. Apply a complete set of stimulus to the two inputs to convince yourself that the gate is working correctly.

Electric includes a powerful tool called a network consistency checker that checks that the schematic and layout are equivalent. This is especially valuable when your design is too complex to verify completely through simulation. The checker relies upon graph theory algorithms. If your layout and schematic match, you have much greater confidence that a bug hasn't crept into your design. If the two don't match, Electric will provide some cues to help you track down the problem. Unfortunately, the hints can be rather cryptic and require a good deal of practice to use effectively.

Use Facets • Edit Facet to be sure both the nand2{sch} and nand2{lay} facets are open. NCC will check the open facet windows against each other and will complain if you don't have exactly two facet windows open. To use NCC, choose Tools • Network • Network Options. In the dialog box, set for all facets to Flatten Hierarchy. Check the Check Export Names and Ignore Power and Ground boxes. Click Clear valid NCC dates to start fresh, then Do NCC now to run the check. If your design is correct, you should get a message of:

```
Comparing facet nand2{sch} (4 components, 6 nets) with facet nand2{lay} (4 components, 6 nets)
- Checking this facet only; Ignoring Power and Ground nets
- Parallel components not merged; Serial transistors not merged
- Checking export names; Ignoring component sizes
Facets nand2{sch} and nand2{lay} are equivalent (0.02 seconds)
```

This message means that the layout and schematic each have four components, i.e. the two NMOS and two PMOS transistors. They also each have six nets: VDD, GND, A, B, Y, and the wire between the series NMOS transistors. NCC finds that the facets are equivalent.

If your design is not equivalent, NCC will complain. One common error is to mislabel the ports. For example, the two NMOS transistors are in series, with one input closer to GND than the other. If the inputs a and b are in opposite order in the layout than in the schematic, you will see the message:

```
******* Export differences have been found! (0.01 seconds)
```

If you press the > key repeatedly to show each error until no more errors exist, you will see the inputs highlighted and see the following messages printed:

```
NCC: Export names 'nand2{sch}:b' and 'nand2{lay}:a' do not match
NCC: Export names 'nand2{sch}:a' and 'nand2{lay}:b' do not match
```

You can confirm the problem is only a matter of names by rerunning NCC with the Check Export Names option deselected. You should get no errors if the topologies are identical between layout and schematic and only the names fail to match. Switch the exports in the layout to fix the problem.

If you get more fundamental errors, you may be able to press the > key and get clues about where NCC is finding mismatches. Often this is hard to interpret, especially when you have a hierarchical design as you will do later. You can also use the Tools • Network • Do NCC Preanalysis command to print a list of networks and components in the design. If you find different numbers of components or networks in any category, you have a hint to where the problem lies. If desperate, you can turn on the Verbose NCC (text) or (graphics) options in the Network options dialog to get hints about how NCC seeks to match components between the layout and schematic. Finally, a manual inspection or simulation may suffice to discover your problem.

5.3. Hierarchical Design

Your next goal is to create a 2-input AND gate. You will build this from your 2-input NAND and your inverter, just as you did in schematics. Therefore, you will first need to create a facet named inv of type layout and draw an inverter like that of Figure 29. Use DRC, ERC, NCC, and simulation to verify your inverter.

Next, create a new layout called and2. Change the technology to mocmos. Instantiate the nand2{lay} and inv{lay} layouts. ALWAYS use the New Facet Instance to create layout from preexisting facets. NEVER build a cell by cutting and pasting entire existing cells. If you do, then make a correction to the original cell, your correction will not propagate to the new layout. However, this does mean that you will need to make all the connections to existing ports in your instantiated cells.

Initially the layouts appear as black boxes with ports. Select both and use the Facet • Expand Facet Instances • All the Way command to view the contents of each layout. Wire together power and ground. Move the cells together as closely as possible without violating design rules. This is easy to do if you designed your cells so power and ground extend two lambda beyond the left and right edges of the cells; you may wish to edit your cells to do this. Connect the output of the nand2 to the input of the inv using a vertical metal2 line to a via to horizontal metal1 to a poly contact and down to the inverter in poly. Remember that connections may only occur between the ports of the two cells.

Electric provides handy short cuts to automatically create vias and contact cuts for you. If you select two layers that could be joined with a via or contact and right click, the tool will automatically insert the appropriate contact for you.

Export the two inputs, the output, and power and ground. The final and gate should resemble Figure 33. Run DRC, ERC, and simulation to verify your design.

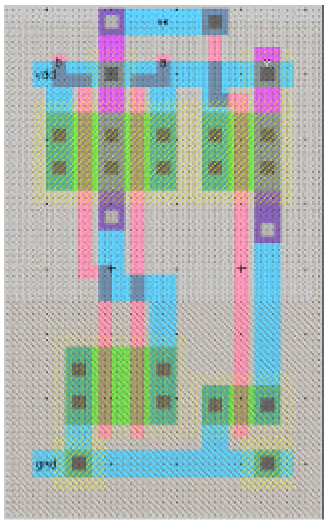


Figure 33: and2{lay}

Finally, do a network compare. NCC has three modes in the Network Options: Check Current Facet Only, Flatten Hierarchy, and Recursively Check Subfacets. Check Current Facet Only assumes all subfacets are perfect. Don't trust this mode unless you have absolute confidence that all the other parts of the design are correct. Flatten Hierarchy smashes your entire design into one giant netlist for its internal comparison. Recursively Check Subfacets checks each subfacet. NCC is a relatively new feature that has received many bug fixes, so it is wise to try both Flatten Hierarchy and Recursively Check Subfacets to catch any problems that one mode misses. In each case, be sure to select Check Export Names and Ignore Power and Ground. Do not set any individual facet overrides. After you check a facet, Electric marks it as clean and thus not in need of rechecking. Press the Clear Valid NCC Dates before each NCC run to ensure NCC will recheck your whole design each time.

Now that your and2 gate is complete, use the Info • Check and Repair Libraries command to look for any inconsistencies in your library. You shouldn't expect any, but Electric has been known to do strange things from time to time, especially in the hands of novice users. Therefore, you may wish to check your library every few hours.

5.4. Other Gates

Draw layout for the following gates. Perform a design rule check and fix any errors. Simulate that they work correctly. Run a network consistency check against the schematic. Finally, do an electrical rule check. Use the following guidelines in your layouts.

?? power and ground run horizontally in Metal 1 on a 60? center-to-center spacing
?? n-well covers all the cell starting 35 ? down from the centerline of VDD
?? all transistors, wires, and well contacts fit between the power and ground lines
?? all transistors should be within 100? of a well contact
?? avoid long routes in diffusion or polysilicon

Cell	Inputs	Outputs
nor2	a, b	y
or2	a, b	y
tristate	a, en, enb	y
latch	d, ph, phb	q

6. Chip Assembly and Tapeout

To assemble a chip, you will need to connect the circuits you have designed to a pad frame. The pad frame has metal bonding pads large enough for a skilled technician with a microscope to attach a gold bonding wire from the pad on your chip to the pin in the package so your chip can communicate with the real world. Then you will carefully check your chip for design errors; fabricating even a simple chip costs $1000 and takes nearly three months mistakes are costly. Finally, you will write your design files in Caltech Interchange Format (CIF) and send them to MOSIS for manufacturing.

6.1. Configuring Technology Options

To tape out targeting the 0.5 ?m AMI process, use the Technology • Technology Options dialog to select submicron rules, 3 metal layers, and alternate active and poly contact rules. Use the Technology • Change Units dialog to set lambda to 600 half-milimicrons (i.e. 0.3 ?m). Thus, transistor gates are drawn with a length of 0.6 ?m and are automatically scaled during manufacturing to 0.5 ?m.

To tape out targeting the 1.5 ?m AMI process, use the Technology • Technology Options dialog to select scmos rules, 2 metal layers, disallow stacked vias, and alternate active and poly contact rules. Use the Technology • Change Units dialog to set lambda to 1600 half-milimicrons (i.e. 0.8 ?m). Thus, transistor gates are drawn with a length of 1.6 ?m and are automatically scaled during manufacturing to 1.5 ?m.

6.2. Pad Frames and Pad Frame Generator

For the 0.5 mm AMI process, use the Electric pad frame generator with a pad frame library developed by David Diaz. A batch of chips has successfully been fabricated with this pad frame library in Spring 2000.

Electric provides a handy pad frame generator that automatically assembles a pad frame for you from a library. To use the pad frame generator, you need your library with a top level layout named core{lay} with exports on its inputs and outputs, a pad library, and a pad arrangement file. The pad library is named ami05diazpads.elib. A sample pad arrangement file for a multiplier is named ami05mul.arr. Look at the pad arrangement file in a text editor. The first part of the file should look lie this:

```
; specify the cell library with the pads
celllibrary ami05diazpads.elib

; create a facet called "padframe"
facet padframe{lay}

; place this facet as the "core"
core core{lay}

; set the alignment of the pads (specifying input and output port names)
align pad_corner{lay}      dvdd-1 dvdd
align pad_in{lay}          dvdd-1 dvdd
align pad_out{lay}         dvdd-1 dvdd
align pad_analog{lay}      dvdd-1 dvdd
align pad_raw{lay}         dvdd-2 dvdd-1
align pad_dvdd{lay}        dvdd-1 dvdd
align pad_dgnd{lay}        dvdd-1 dvdd

;; replace the pad_in and pad_out statements with the pad frame arrangement you want.
;; keep at least two pad_dvdd and pad_dgnd pads somewhere on the chip to supply
;; power and ground to the pads and core

; place the top edge of pads, starting with upper-right
place pad_corner{lay}
place pad_dvdd{lay}
place pad_in{lay} din=A0
place pad_in{lay} din=A1
place pad_in{lay} din=A2
place pad_in{lay} din=A3
place pad_dgnd{lay}
place pad_in{lay} din=A4
place pad_in{lay} din=A5
place pad_in{lay} din=A6
place pad_in{lay} din=A7

; place the left edge of pads
rotate cc
place pad_corner{lay}
place pad_dvdd{lay}
place pad_out{lay} dout=B0
…
```

Modify the pad file by changing the pad_in and pad_out statements to connect the appropriate types of pads to the core exports. Be sure to keep at least three pad_dvdd (dirty VDD) and three pad_dgnd pads. Output pads create huge current spikes as the large inverters switch and charge up the enormous off-chip capacitances. The bond wires have inductance, so the current spikes cause voltage drops across the inductors. The dirty notation indicates that the same pad is used to supply current to the noisy or *dirty* output pads as to the quiet or *clean* core.

Use the Tools • Generation • Pad Frame command. This will create a new facet called padframe{lay} with generic unrouted arcs connecting the pads to the core. Inspect the padframe to be sure it looks reasonable.

Now you must connect power and ground for the pads together. Each pad has a VDD and a GND export on both edges that must be connected to the adjacent pad. To do this, we can take advantage of the mimic stitching capability to save having to make every connection by hand. Choose Tools • Routing • Routing Options and be sure all options are unchecked to let the mimic stitcher make as many connections as it can deduce. Then left click on the vdd export on the edge of one of the pads and right click on the corresponding export on the adjacent pad to make a very short connection (the exports should be overlapping). Choose Tools • Routing • Mimic Stitch Now to ask Electric to wire together all connections that it thinks are similar. Often it will miss some; repeat the wiring and mimic stitching until you've made all the vdd and gnd connections. You may wish to run DRC to ensure you found all the connections.

Finally, you must convert the generic unrouted arcs to real layout. You can use the river router to automate this tedious task. This is not yet working well.

For the 1.5 mm AMI process, use a pad frame provided by MOSIS. This pad frame has been imported into Electric as pure layer nodes and thus is not easy to verify. A batch of chips has been fabricated with these pads from Electric in Fall 2000. Ring oscillators operated correctly, suggesting that the pads work. However, the core of the chip does not operate. To use this pad frame you must hook up the core to the pads by hand. Be sure to set the enable signals on each pad appropriately for input, output, or bi-directional operation.

6.3. Pretapeout Checks

Do a thorough check on the top-level layout using the following steps and fix any errors:

1. Tools • DRC • DRC Options: Clear valid DRC dates
   Note: For 1.5 micron process, it is wise to satisfy both SCMOS and SUBMICRON rules so the design can port to the 0.5 micron process more easily.
2. Tools • DRC • Hierarchical Check
3. Tools • Network • Network Options:
   Clear valid NCC Dates
   Recursively Check Subfacets
   Select Ignore Power and Ground and Check Export Names
   Do NCC Now
4. Tools • Electrical Rules • Analyze Wells

Simulate your design. Recommend simulators include IRSIM (built-in), Verilog, and SPICE. Verilog and SPICE simulation are presently beyond the scope of this tutorial.

6.4. CIF Generation and Interpretation

To write a CIF file:

1. File • IO Options • CIF Options:
   Output Instantiates Top Level
   Select Report Resolution Errors
   Output resolution 0.25 (due to an Electric bug; should be 0.5 when fixed)

2. File • Export • CIF

Electric may issue warnings about obscured facet exports; these may be ignored. However, if you receive any resolution errors, be sure to fix them. These occur if the elements of the design are not all on a 0.5 lambda grid and usually indicates sloppy layout practices such as not using facet centers or not drawing on grid.

Electric will report a MOSIS Cyclical Redundancy Check (CRC) code to ensure your CIF transmits correctly. Record the two numbers indicating checksum and count, respectively.

6.5. MOSIS Tapeout Instructions

MOSIS is a service that combines designs from various sources and buys space on a manufacturing line. Using MOSIS is more cost-effective than buying space directly from a vendor for small orders because the mask and wafer costs can be split among many projects. The Semiconductor Industry Association and a number of vendors donate funding to support fabricating chips for academic projects.

Taping out to MOSIS involves completing two web forms: the New Project request and the Fabricate request. The MOSIS web forms are at:

https://www.mosis.org/Webforms/menu-webforms.html

This section describes how to complete the form for 1.5 and 0.5 micron AMI 40-pin Tiny Chips. Start with the New Project Form:

	AMI 0.5 ?m	AMI 1.5 ?m
Account Number	<institution dependent>	
Account Password	<institution dependent>	
Design Name	<project dependent>	
Design Password (& retype)	<project dependent>	
Net Address	<your email>	
Run Type	Shared multi-project run	
Fill Authorized	Yes (unless you are building inductors)	
Phone Number	<your phone>	
Technology Code	SCN3ME _SUBM	SCNE
Lambda	0.3	0.8
Foundry	AMI	AMI
Pad Count	40	40
Design Size X	1500	2200
Design Size Y	1500	2200
Routing Label	<your name>	

Record the 5-digit design number returned from the form, then use the Fabricate form:

Design Number	<returned from New Project form>
Design Password	<specified on New Project form>
Layout Format	CIF
Compression	Uncompressed
Checksum Type	CRC
Checksum	<recorded when generating CIF>
Count	<recorded when generating CIF>
FTP Send Host	Your host (e.g. davidii.eng.hmc.edu)
FTP Send Password (& retype)	asillypassword
FTP Send Filename	<CIF filename>

7. Tips and Tricks

Arraying
Mimic Stitching
Silicon Compiler

Update switch-level simulation when IRSIM interface is done
Add verilog and spice simulation
Update 6.4 when CIF netlist bug is fixed